ESCUELA INTERNACIONAL POSGRADOS

Máster Avanzado de Programación en Python para Hacking, BigData y Machine Learning

CERTIFICACIÓN PCAP

LECCIÓN 2

Tipos de datos, operadores, estructuras de control y funciones

ÍNDICE

INTRODUCCIÓN

En esta lección repasaremos los aspectos fundamentales de los literales, las variables, los operadores y expresiones, las estructuras de control y las funciones, especialmente aquellos que considero, desde mi punto de vista personal, muy recurrentes en las preguntas del PCAP.

OBJETIVOS

Al finalizar esta lección serás capaz de:

1. Conocer los diferentes tipos de literales de Python
2. Conocer los distintos operadores y su jerarquía
3. Conocer las diferentes estructuras condicionales y bucles
4. Conocer como definir y llamar a una función y el alcance d ellos nombres dentro de ellas

Literales de Python

LITERAL: Se refiere a datos cuyos valores están determinados por el mismo literal, es decir, son una notación para representar un valor fijo en el código fuente.

Los literales numéricos:
- Enteros
- Flotantes

Las cadenas de caracteres

Los valores Booleanos

El literal None

Operadores y Expresiones

OPERADOR: Símbolo del lenguaje de programación capaz de realizar operaciones con los valores.

EXPRESIÓN: Combinación de datos y operadores.

- Operadores aritméticos
- Operadores de comparación
- Operadores lógicos
- Operadores de asignación

Operadores y Expresiones: Operadores aritméticos

Los operadores aritméticos son:

- Suma: a + b
- Resta: a – b
- Multiplicación: a * b
- División: a / b
- División entera: a // b
- Exponenciación: a ** b
- Residuo o módulo: a % b

Importante

- +,-,*,//,**,% Si unos de los operandos es flotante el resultados también lo será. Sino será entero.
- / El resultado será siempre un flotante
- // Devuelve la división redondeada al entero inferior más cercano. Cuidado con los negativos.

Operadores y Expresiones: Operadores de comparación

Los operadores de comparación más utilizados son:

- Igual: ==
- Mayor que: >
- Menor que: <
- Mayor o igual que: >=
- Menor o igual que: <=
- Distinto que: !=

Importante

- Estas operaciones devuelven un valor booleano.

- 2.0 == 2 -> True
- “2” == 2 -> False

Operadores y Expresiones: Operadores lógicos

Los operadores lógicos son:

- Conjunción: and
- Disyunción: or
- Negación: not

Importante

- Estas operaciones trabajan con valores booleanos y devuelven un valor booleano.

Operadores y Expresiones: Operadores de comparación bit a bit (bitwise)

Los operadores de bit a bit son:

- Conjunción: &
- Disyunción: |
- Disyunción exclusiva o xor: ^
- Negación: ~

Importante

- Estas operaciones trabajan con bits individuales, así como con enteros (los convierte en binario).
- Con valores flotantes devolverá error.
- True es internamente convertido a 1 y False a 0

Operadores y Expresiones: Operadores de asignación

Los operadores de asignación son:

- Asignación simple: =
- Asignación compuesta:
  - +=
  - -=
  - *=
  - /=
  - %=
  - **=
  - //=
  - &=
  - |=
  - ^=

Importante

- x += y -> x = x + y
- x += y – 1 -> x= x + (y – 1)

Operadores y Expresiones: Jerarquía de prioridades de operadores

Importante

- Enlace: Orden en que los operadores misma prioridad actúan.
- Todos -> De derecha a izquierda.
- ** -> Excepción de izquierda a derecha.

operadores	descripción
**	Exponenciación (prioridad más alta)
~, +, -	Negación bit a bit, más y menos **unario**
*, /, //, %	Multiplicación, división, división entera y módulo
+, -	Suma Resta
>> , <<	Desplazamiento de bits
&	Conjunción bit a bit
^ |	Disyuntiva y xor bit a bit
<=, <=, >>	Comparación
==, ! =	Operador de igualdad
=, +=, -=, /=, //=, %=, * =, **=	Operadores de asignación

Variables en Python

Una variable es una ubicación nombrada reservada para almacenar valores en la memoria.

Está compuesta por un nombre y un valor.

Importante

- Las variables se crean cuando se les asigna un valor.
- Si se intenta usar una que no existe -> NameError

Reglas que deben seguir los nombres:

- Compuesto por MAYÚSCULAS, minúsculas, dígitos o "_"
- Debe comenzar por letra o "_", pero no por dígito.
- Las mayúsculas y minúsculas se tratan de forma distinta.
- No hay restricciones de longitud.
- No se puede utilizar una palabra reservada:

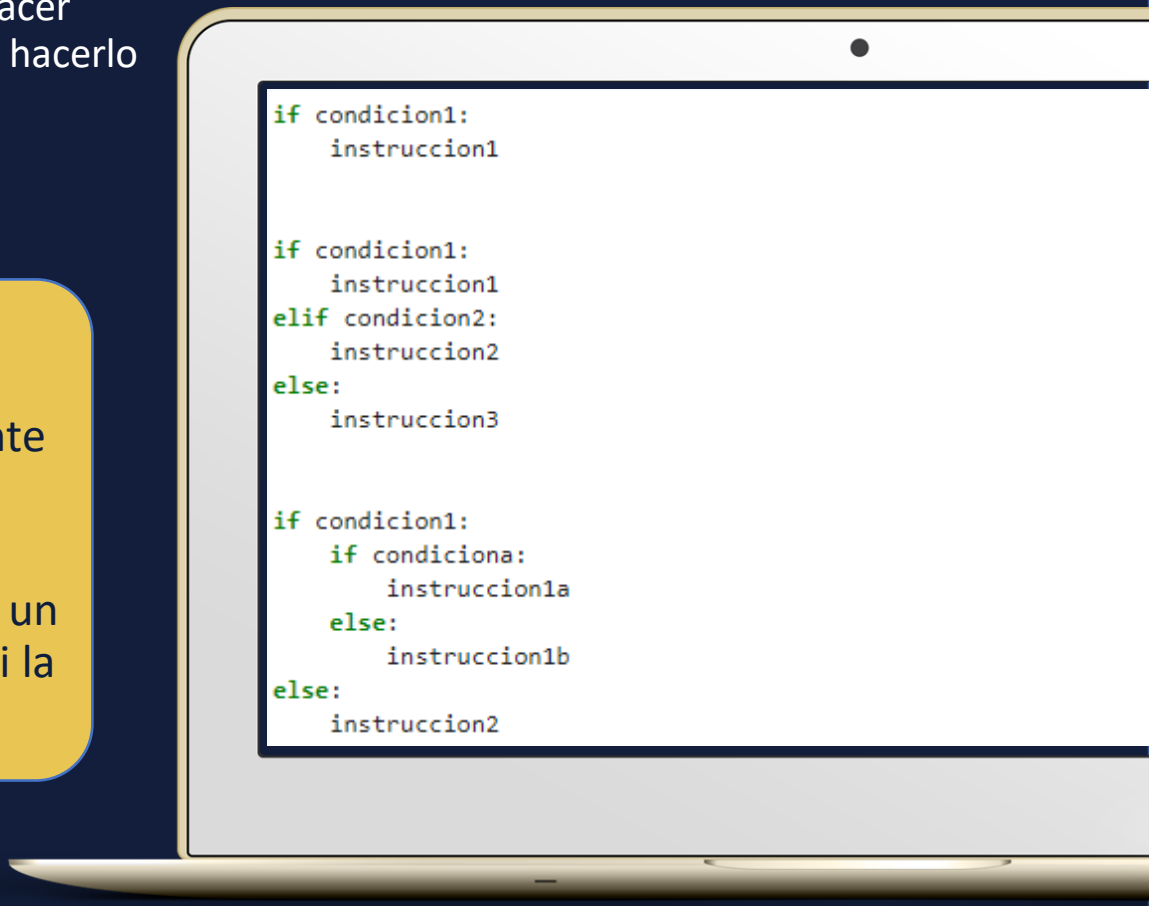
```
'False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del',
'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield'
```

Estructuras de control: Condicionales (if)

La instrucción condicional permite hacer algo si se cumple una condición y no hacerlo si no se cumple.

Importante

- La sangría es muy importante (tabulador o 4 espacios, siempre lo mismo)
- Si la condición no devuelve un booleano, se comprobará si la variable no está vacía.



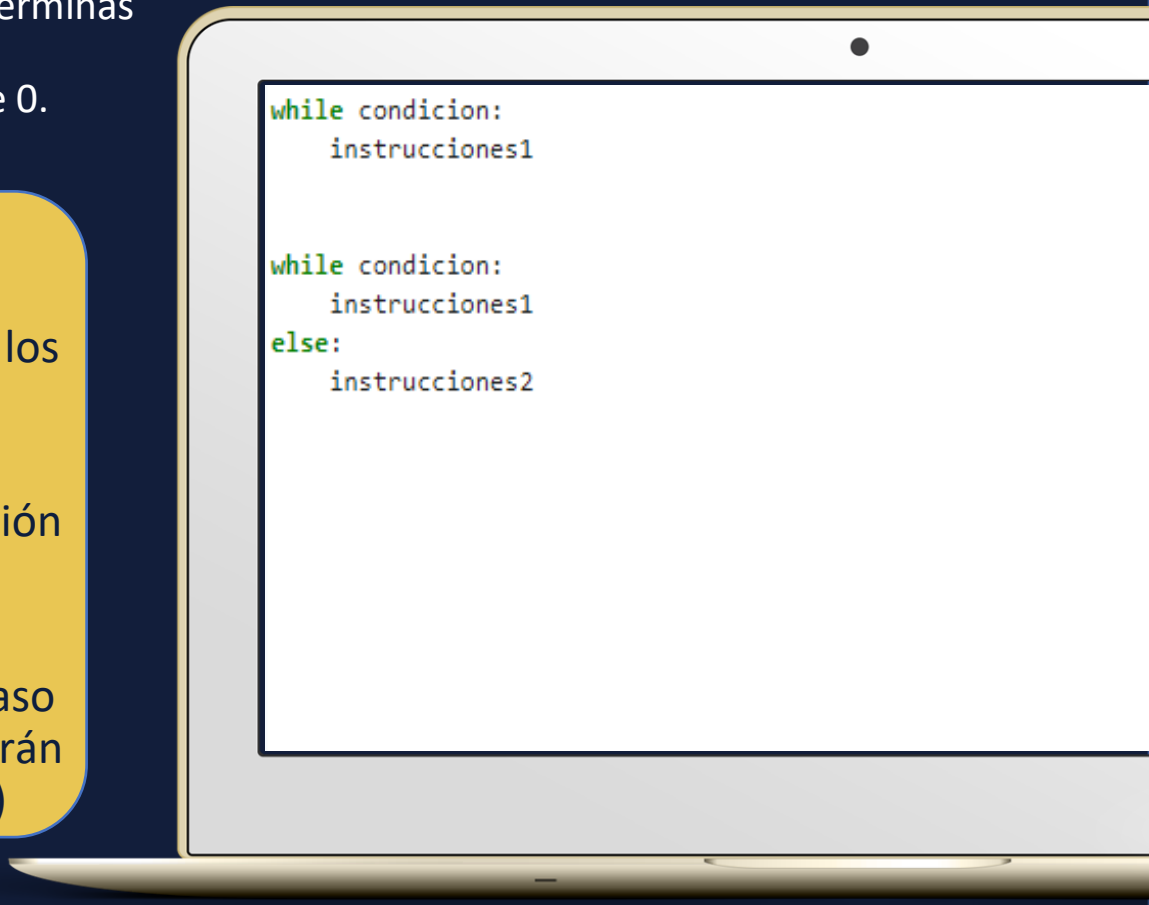
```
if condicion1:
    instruccion1


if condicion1:
    instruccion1
elif condicion2:
    instruccion2
else:
    instruccion3


if condicion1:
    if condiciona:
        instruccion1a
    else:
        instruccion1b
else:
    instruccion2
```

Estructuras de control: Bucles (while)

Un bucle while permite ejecutar determinas instrucciones mientras la condición evaluada sea verdadera o distinta de 0.

Importante

- Hay que tener cuidado con los bucles infinitos.
- Recordad que el bucle terminará cuando la condición es False o 0.
- Se puede parar con la instrucción break (en ese caso si hay un else no se ejecutarán las instrucciones de dentro)



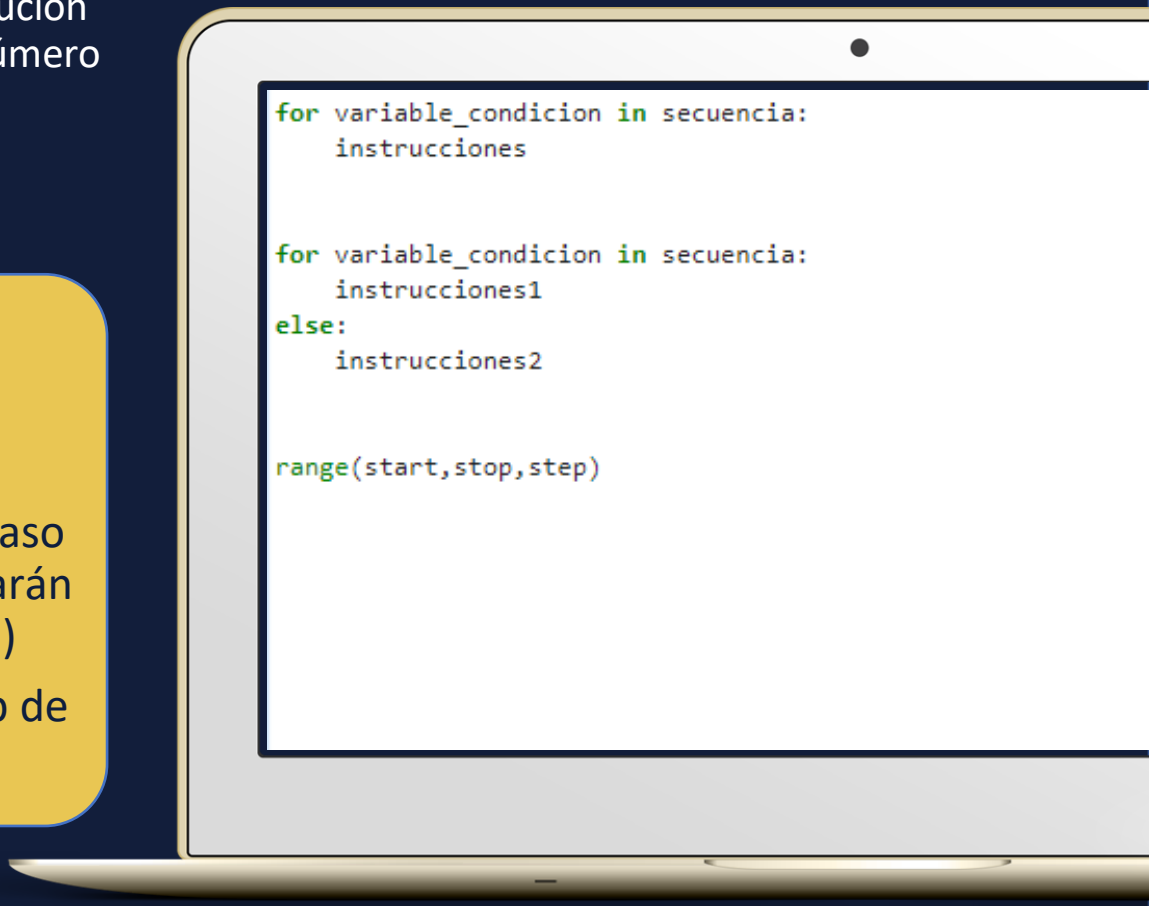
```
while condicion:
    instrucciones1


while condicion:
    instrucciones1
else:
    instrucciones2
```

Estructuras de control: Bucles (for)

Un bucle for permite hacer una ejecución repetitiva pero controlada con un número definido de ciclos.

Importante

- Se puede parar con la instrucción break (en ese caso si hay un else no se ejecutarán las instrucciones de dentro)
- Los parámetros start y step de range() son opcionales



```
for variable_condicion in secuencia:
    instrucciones


for variable_condicion in secuencia:
    instrucciones1
else:
    instrucciones2


range(start,stop,step)
```

Funciones

Una función es una pieza de código que recibe unos parámetros de entrada y entrega uno resultado de salida.

Surgen por la necesidad de reutilizar código.

Importante

- El nombre sigue las mismas normas que los nombres de variables.
- Puede no tener parámetros o la sentencia return

```
def nombre_funcion1(parametro1, parametro2,...):
    cuerpoFuncion
    return resultado
nombre_funcion1(parametro1, parametro2,...)


def nombre_funcion2(parametro1, parametro2,...):
    cuerpoFuncion
    return
nombre_funcion2(parametro1, parametro2,...)


def nombre_funcion3(parametro1, parametro2=0):
    cuerpoFuncion
    return resultado
nombre_funcion3(parametro1)
```

Funciones: Métodos de paso de parámetros

Si el número de argumentos es incorrecto error -> TypeError
Si se intenta llamar a una función antes de que se haya definido error -> NameError

Paso de parámetros posicionales

```
def nombre_funcion1(param1, param2):
    cuerpoFuncion
    return resultado


nombre_funcion1(valor1, valor2)
```

Paso de argumentos con palabras clave

```
def nombre_funcion1(param1, param2):
    cuerpoFuncion
    return resultado


nombre_funcion1(param1=1, param2=2)
```

Combinación de ambos pasos

```
def nombre_funcion1(param1, param2):
    cuerpoFuncion
    return resultado


nombre_funcion1(1, param2=2)
```

Los parámetros posicionales siempre deberán ir al inicio

Funciones: Alcance de los nombres o Scopes

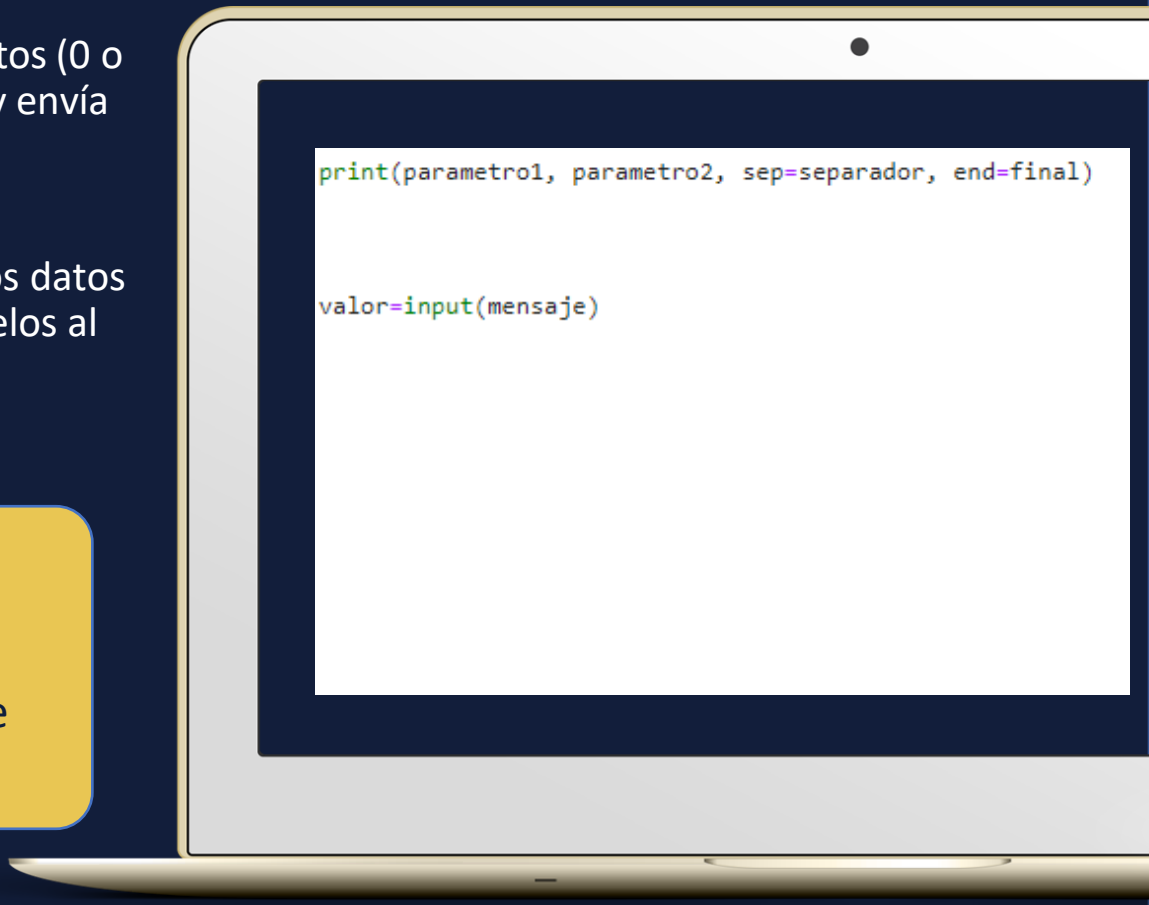
El alcance de un nombre (por ejemplo, el de una variable se define como la parte del código donde el nombre es reconocido correctamente.

- Una variable definida fuera de una función tiene alcance dentro del cuerpo de la función, es decir, la función podrá leer su valor. Excepción: aquellas que tienen el mismo nombre.
- Si la variable es de tipo escalar dentro no se podrá modificar su valor. Para modificarlo hay que usar global.
- Si la variable es una lista, se podrá modificar su valor.
- Una variable definida dentro de una función sólo tiene alcance dentro de esa función.

Funciones: Las funciones print() e input()

La función print() toma los argumentos (0 o n), los convierte en formato legible y envía los datos al dispositivo de salida.

La función input() es capaz de leer los datos introducidos por el usuario y pasárselos al programa en ejecución.



Importante

- La función input() devuelve una cadena de caracteres.

CONCLUSIONES

1. Hemos visto los diferentes tipos de literales y operadores
2. Hemos visto las estructuras condicionales y bucles
3. Hemos visto cómo definir y llamar a una función, así como el scope de las variables

MUCHAS GRACIAS POR SU ATENCIÓN

eip ESCUELA INTERNACIONAL POSGRADOS

tcivera@grupomainjobs.com

Tamara Civera Lorenzo
es.linkedin.com/in/tamara-civera-lorenzo-95962147

twitter.com/eiposgrados

facebook.com/eiposgrados

instagram.com/eiposgrados